Lecture 3: More Word Vectors

Christopher Manning and Richard Socher
Organization. See Calendar

Coding Session!
Wednesday, 6:30 - 9:30pm at Charles B. Thornton Center for Engineering and Management, room 110

Career Fair
Wednesday 11am - 4pm

My first project advice office hour is today
A Deeper Look at Word Vectors

• Finish word2vec

• What does word2vec capture?

• How could we capture this essence more effectively?

• How can we analyze word vectors?
Review: Main ideas of word2vec

• Go through each word of the whole corpus

• Predict surrounding words of each (window’s center) word

\[ p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \]

• Then take gradients at each such window for SGD
Stochastic gradients with word vectors!

- But in each window, we only have at most $2m + 1$ words, so $\nabla_{\theta} J_t(\theta)$ is very sparse!

\[
\nabla_{\theta} J_t(\theta) = \begin{bmatrix}
0 \\
\vdots \\
0 & \nabla_{v_{like}} \\
\vdots \\
0 & \nabla_{u_I} \\
\vdots \\
0 & \nabla_{u_{learning}} \\
\end{bmatrix} \in \mathbb{R}^{2dV}
\]
Stochastic gradients with word vectors!

• We may as well only update the word vectors that actually appear!

• Solution: either you need sparse matrix update operations to only update certain columns of full embedding matrices $U$ and $V$, or you need to keep around a hash for word vectors

$$d \left[ \begin{array}{ccc} \cdots & \cdots & |V| \\ \cdots & \cdots & \cdots \end{array} \right]$$

• If you have millions of word vectors and do distributed computing, it is important to not have to send gigantic updates around!
Approximations: Assignment 1

• The normalization factor is too computationally expensive.

\[ p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)} \]

• Hence, in Assignment 1, you will implement the skip-gram model with **negative sampling**

• Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus a couple of noise pairs (the center word paired with a random word)
Ass 1: The skip-gram model and negative sampling

- From paper: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
- Overall objective function: \( J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta) \)

\[
J_t(\theta) = \log \sigma \left( u_o^T v_c \right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[ \log \sigma \left( -u_j^T v_c \right) \right]
\]

- Where \( k \) is the number of negative samples and we use,
- The sigmoid function! \( \sigma(x) = \frac{1}{1+e^{-x}} \)
  (we’ll become good friends soon)
- So we maximize the probability of two words co-occurring in first log
Ass 1: The skip-gram model and negative sampling

- Slightly clearer notation:

\[ J_t(\theta) = \log \sigma (u_o^T v_c) + \sum_{j \sim P(w)} \left[ \log \sigma (-u_j^T v_c) \right] \]

- Maximize probability that real outside word appears, minimize prob. that random words appear around center word

- \( P(w) = U(w)^{3/4}/Z \),
  the unigram distribution \( U(w) \) raised to the 3/4 power
  (We provide this function in the starter code).
- The power makes less frequent words be sampled more often
Ass 1: The continuous bag of words model

- Main idea for continuous bag of words (CBOW): Predict center word from sum of surrounding word vectors instead of predicting surrounding single words from center word as in skip-gram model.

- To make assignment slightly easier:

  Implementation of the CBOW model is not required (you can do it for a couple of bonus points!), but you do have to do the written problem on CBOW.
Word2vec improves objective function by putting similar words nearby in space.
Summary of word2vec

• Go through each word of the whole corpus
• Predict surrounding words of each word

• This captures cooccurrence of words one at a time

• Why not capture cooccurrence **counts** directly?
Yes we can!

With a co-occurrence matrix X

- 2 options: full document vs. windows
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”

- Instead: Similar to word2vec, use window around each word → captures both syntactic (POS) and semantic information
Example: Window based co-occurrence matrix

- Window length 1 (more common: 5 - 10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.
Window based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

<table>
<thead>
<tr>
<th>counts</th>
<th>I</th>
<th>like</th>
<th>enjoy</th>
<th>deep</th>
<th>learning</th>
<th>NLP</th>
<th>flying</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>like</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>enjoy</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>deep</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>learning</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NLP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>flying</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Problems with simple co-occurrence vectors

Increase in size with vocabulary

Very high dimensional: require a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust
Solution: Low dimensional vectors

- Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector

- Usually 25 – 1000 dimensions, similar to word2vec

- How to reduce the dimensionality?
Method 1: Dimensionality Reduction on $X$

Singular Value Decomposition of co-occurrence matrix $X$.

$$X = \hat{U}US\hat{V}^T$$

$\hat{X}$ is the best rank $k$ approximation to $X$, in terms of least squares.
Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

```python
import numpy as np
la = np.linalg

words = ["I", "like", "enjoy", 
         "deep","learnig","NLP","flying","." ]

X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```
Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values

```python
for i in xrange(len(words)):
    plt.text(U[i, 0], U[i, 1], words[i])
```
Hacks to X

• Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
  • \( \min(X,t) \), with \( t \sim 100 \)
  • Ignore them all
• Ramped windows that count closer words more
• Use Pearson correlations instead of counts, then set negative values to 0
• +++
Interesting semantic patters emerge in the vectors

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005
Interesting syntactic patterns emerge in the vectors

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005
Interesting semantic patterns emerge in the vectors

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005
Problems with SVD

Computational cost scales quadratically for n x m matrix:

$O(mn^2)$ flops (when n<m)

→ Bad for millions of words or documents

Hard to incorporate new words or documents

Different learning regime than other DL models
Count based vs direct prediction

LSA, HAL (Lund & Burgess), COALS (Rohde et al), Hellinger-PCA (Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

NNLM, HLBL, RNN, Skip-gram/CBOW, (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity
Combining the best of both worlds: GloVe

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

• Fast training
• Scalable to huge corpora
• Good performance even with small corpus, and small vectors
What to do with the two sets of vectors?

• We end up with $U$ and $V$ from all the vectors $u$ and $v$ (in columns)

• Both capture similar co-occurrence information. It turns out, the best solution is to simply sum them up:

$$X_{\text{final}} = U + V$$

• One of many hyperparameters explored in *GloVe: Global Vectors for Word Representation* (Pennington et al. (2014))
Glove results

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

Richard Socher
1/17/17
Linear Algebraic Structure of Word Senses, with Applications to Polysemy

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, Andrej Risteski

Presented by: Arun Chaganty
Word vectors encode similarity.
What about polysemy?
1. Polysemous vectors are superpositioned.
2. Senses can be recovered by sparse coding

\[ v = \sum_{i=0}^{D} \alpha_i A_i + \eta \]

- Context vectors (~2000)
- Word vector
- Selectors (< 5)
- Noise
2. Senses can be recovered by sparse coding

<table>
<thead>
<tr>
<th>tie</th>
<th>trousers</th>
<th>blouse</th>
<th>waistcoat</th>
<th>skirt</th>
<th>sleeved pants</th>
</tr>
</thead>
<tbody>
<tr>
<td>season</td>
<td>teams</td>
<td>winning</td>
<td>league</td>
<td>finished</td>
<td>championship</td>
</tr>
<tr>
<td>scoreline</td>
<td>goalless</td>
<td>equaliser</td>
<td>clinching</td>
<td>scoreless</td>
<td>replay</td>
</tr>
<tr>
<td>wires</td>
<td>cables</td>
<td>wiring</td>
<td>electrical</td>
<td>wire</td>
<td>cable</td>
</tr>
<tr>
<td>operatic</td>
<td>soprano</td>
<td>mezzo</td>
<td>contralto</td>
<td>baritone</td>
<td>coloratura</td>
</tr>
</tbody>
</table>
3. Senses recovered are non-native English level

tie

1. Trousers, blouse, pants
2. Season, teams, winning
3. Computer, mouse, keyboard
4. Bulb, light, flash
Summary

Word vectors can capture polysemy!

Word vectors are linear superposition of each sense vector.

Sense-context vectors can be recovered by sparse coding.

The senses recovered are about as good as a non-native English speakers!

Thanks!
How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy → Winning!
Intrinsic word vector evaluation

- **Word Vector Analogies**

  \[ a:b :: c:? \]

  man:woman :: king:?  

  \[ d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||} \]

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions

- Discarding the input words from the search!

- Problem: What if the information is there but not linear?
Glove Visualizations: Company - CEO

- Caterpillar
- Chrysler
- United
- Exxon
- Wal-Mart
- IBM
- Citigroup
- Viacom
- Verizon
- Vodafone
- Oberhelman
- Marchionne
- Smisek
- Tillerson
- McMillon
- Corbat
- Rometty
- Dauman
- McAdam
- Colao
Glove Visualizations: Superlatives
Other fun word2vec analogies

<table>
<thead>
<tr>
<th>Expression</th>
<th>Nearest token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris - France + Italy</td>
<td>Rome</td>
</tr>
<tr>
<td>bigger - big + cold</td>
<td>colder</td>
</tr>
<tr>
<td>sushi - Japan + Germany</td>
<td>bratwurst</td>
</tr>
<tr>
<td>Cu - copper + gold</td>
<td>Au</td>
</tr>
<tr>
<td>Windows - Microsoft + Google</td>
<td>Android</td>
</tr>
<tr>
<td>Montreal Canadiens - Montreal + Toronto</td>
<td>Toronto Maple Leafs</td>
</tr>
</tbody>
</table>
Details of intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and Semantic examples from http://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt

: city-in-state
Chicago Illinois Houston Texas
Chicago Illinois Philadelphia Pennsylvania
Chicago Illinois Phoenix Arizona
Chicago Illinois Dallas Texas
Chicago Illinois Jacksonville Florida
Chicago Illinois Indianapolis Indiana
Chicago Illinois Austin Texas
Chicago Illinois Detroit Michigan
Chicago Illinois Memphis Tennessee
Chicago Illinois Boston Massachusetts

problem: different cities may have same name
Details of intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and **Semantic** examples from

: capital-world

Abuja Nigeria Accra Ghana
Abuja Nigeria Algiers Algeria
Abuja Nigeria Amman Jordan
Abuja Nigeria Ankara Turkey
Abuja Nigeria Antananarivo Madagascar
Abuja Nigeria Apia Samoa
Abuja Nigeria Ashgabat Turkmenistan
Abuja Nigeria Asmara Eritrea
Abuja Nigeria Astana Kazakhstan
Details of intrinsic word vector evaluation

- Word Vector Analogies: **Syntactic** and Semantic examples from gram4-superlative
  
  bad worst big biggest
  bad worst bright brightest
  bad worst cold coldest
  bad worst cool coolest
  bad worst dark darkest
  bad worst easy easiest
  bad worst fast fastest
  bad worst good best
  bad worst great greatest
Details of intrinsic word vector evaluation

- Word Vector Analogies: **Syntactic** and Semantic examples from

: gram7-past-tense
- dancing danced decreasing decreased
dancing danced describing described
dancing danced enhancing enhanced
dancing danced falling fell
dancing danced feeding fed
dancing danced flying flew
dancing danced generating generated
dancing danced going went
dancing danced hiding hid
dancing danced hitting hit
Analogy evaluation and hyperparameters

- Very careful analysis: Glove word vectors

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ivLBL</td>
<td>100</td>
<td>1.5B</td>
<td>55.9</td>
<td>50.1</td>
<td>53.2</td>
</tr>
<tr>
<td>HPCA</td>
<td>100</td>
<td>1.6B</td>
<td>4.2</td>
<td>16.4</td>
<td>10.8</td>
</tr>
<tr>
<td>GloVe</td>
<td>100</td>
<td>1.6B</td>
<td>67.5</td>
<td>54.3</td>
<td>60.3</td>
</tr>
<tr>
<td>SG</td>
<td>300</td>
<td>1B</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>CBOW</td>
<td>300</td>
<td>1.6B</td>
<td>16.1</td>
<td>52.6</td>
<td>36.1</td>
</tr>
<tr>
<td>vLBL</td>
<td>300</td>
<td>1.5B</td>
<td>54.2</td>
<td>64.8</td>
<td>60.0</td>
</tr>
<tr>
<td>ivLBL</td>
<td>300</td>
<td>1.5B</td>
<td>65.2</td>
<td>63.0</td>
<td>64.0</td>
</tr>
<tr>
<td>GloVe</td>
<td>300</td>
<td>1.6B</td>
<td>80.8</td>
<td>61.5</td>
<td>70.3</td>
</tr>
<tr>
<td>SVD</td>
<td>300</td>
<td>6B</td>
<td>6.3</td>
<td>8.1</td>
<td>7.3</td>
</tr>
<tr>
<td>SVD-S</td>
<td>300</td>
<td>6B</td>
<td>36.7</td>
<td>46.6</td>
<td>42.1</td>
</tr>
<tr>
<td>SVD-L</td>
<td>300</td>
<td>6B</td>
<td>56.6</td>
<td>63.0</td>
<td>60.1</td>
</tr>
<tr>
<td>CBOW†</td>
<td>300</td>
<td>6B</td>
<td>63.6</td>
<td>67.4</td>
<td>65.7</td>
</tr>
<tr>
<td>SG†</td>
<td>300</td>
<td>6B</td>
<td>73.0</td>
<td>66.0</td>
<td>69.1</td>
</tr>
<tr>
<td>GloVe</td>
<td>300</td>
<td>6B</td>
<td>77.4</td>
<td>67.0</td>
<td>71.7</td>
</tr>
<tr>
<td>CBOW</td>
<td>1000</td>
<td>6B</td>
<td>57.3</td>
<td>68.9</td>
<td>63.7</td>
</tr>
<tr>
<td>SG</td>
<td>1000</td>
<td>6B</td>
<td>66.1</td>
<td>65.1</td>
<td>65.6</td>
</tr>
<tr>
<td>SVD-L</td>
<td>300</td>
<td>42B</td>
<td>38.4</td>
<td>58.2</td>
<td>49.2</td>
</tr>
<tr>
<td>GloVe</td>
<td>300</td>
<td>42B</td>
<td>81.9</td>
<td>69.3</td>
<td>75.0</td>
</tr>
</tbody>
</table>
Analogy evaluation and hyperparameters

- Asymmetric context (only words to the left) are not as good

  (a) Symmetric context  
  (b) Symmetric context  
  (c) Asymmetric context

- Best dimensions ~300, slight drop-off afterwards
- But this might be different for downstream tasks!

- Window size of 8 around each center word is good for Glove vectors
Analogy evaluation and hyperparameters

- More training time helps
More data helps, Wikipedia is better than news text!
Another intrinsic word vector evaluation

- Word vector distances and their correlation with human judgments
- Example dataset: WordSim353
  
  [Link to dataset](http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/)

<table>
<thead>
<tr>
<th>Word 1</th>
<th>Word 2</th>
<th>Human (mean)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tiger</td>
<td>cat</td>
<td>7.35</td>
</tr>
<tr>
<td>tiger</td>
<td>tiger</td>
<td>10.00</td>
</tr>
<tr>
<td>book</td>
<td>paper</td>
<td>7.46</td>
</tr>
<tr>
<td>computer</td>
<td>internet</td>
<td>7.58</td>
</tr>
<tr>
<td>plane</td>
<td>car</td>
<td>5.77</td>
</tr>
<tr>
<td>professor</td>
<td>doctor</td>
<td>6.62</td>
</tr>
<tr>
<td>stock</td>
<td>phone</td>
<td>1.62</td>
</tr>
<tr>
<td>stock</td>
<td>CD</td>
<td>1.31</td>
</tr>
<tr>
<td>stock</td>
<td>jaguar</td>
<td>0.92</td>
</tr>
</tbody>
</table>
### Closest words to “Sweden” (cosine similarity)

<table>
<thead>
<tr>
<th>Word</th>
<th>Cosine distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>norway</td>
<td>0.760124</td>
</tr>
<tr>
<td>denmark</td>
<td>0.715460</td>
</tr>
<tr>
<td>finland</td>
<td>0.620022</td>
</tr>
<tr>
<td>switzerland</td>
<td>0.588132</td>
</tr>
<tr>
<td>belgium</td>
<td>0.585835</td>
</tr>
<tr>
<td>netherlands</td>
<td>0.574631</td>
</tr>
<tr>
<td>iceland</td>
<td>0.562368</td>
</tr>
<tr>
<td>estonia</td>
<td>0.547621</td>
</tr>
<tr>
<td>slovenia</td>
<td>0.531408</td>
</tr>
</tbody>
</table>
Correlation evaluation

- Word vector distances and their correlation with human judgments

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>WS353</th>
<th>MC</th>
<th>RG</th>
<th>SCWS</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVD</td>
<td>6B</td>
<td>35.3</td>
<td>35.1</td>
<td>42.5</td>
<td>38.3</td>
<td>25.6</td>
</tr>
<tr>
<td>SVD-S</td>
<td>6B</td>
<td>56.5</td>
<td>71.5</td>
<td>71.0</td>
<td>53.6</td>
<td>34.7</td>
</tr>
<tr>
<td>SVD-L</td>
<td>6B</td>
<td>65.7</td>
<td>72.7</td>
<td>75.1</td>
<td>56.5</td>
<td>37.0</td>
</tr>
<tr>
<td>CBOW†</td>
<td>6B</td>
<td>57.2</td>
<td>65.6</td>
<td>68.2</td>
<td>57.0</td>
<td>32.5</td>
</tr>
<tr>
<td>SG†</td>
<td>6B</td>
<td>62.8</td>
<td>65.2</td>
<td>69.7</td>
<td>58.1</td>
<td>37.2</td>
</tr>
<tr>
<td>GloVe</td>
<td>6B</td>
<td>65.8</td>
<td>72.7</td>
<td>77.8</td>
<td>53.9</td>
<td>38.1</td>
</tr>
<tr>
<td>SVD-L</td>
<td>42B</td>
<td>74.0</td>
<td>76.4</td>
<td>74.1</td>
<td>58.3</td>
<td>39.9</td>
</tr>
<tr>
<td>GloVe</td>
<td>42B</td>
<td><strong>75.9</strong></td>
<td><strong>83.6</strong></td>
<td><strong>82.9</strong></td>
<td><strong>59.6</strong></td>
<td><strong>47.8</strong></td>
</tr>
<tr>
<td>CBOW*</td>
<td>100B</td>
<td>68.4</td>
<td>79.6</td>
<td>75.4</td>
<td>59.4</td>
<td>45.5</td>
</tr>
</tbody>
</table>

- Some ideas from Glove paper have been shown to improve skip-gram (SG) model also (e.g. sum both vectors)
But what about ambiguity?

• You may hope that one vector captures both kinds of information (run = verb and noun) but then vector is pulled in different directions

• Alternative described in: *Improving Word Representations Via Global Context And Multiple Word Prototypes* (Huang et al. 2012)

• Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters bank₁, bank₂, etc
But what about ambiguity?

- Improving Word Representations Via Global Context And Multiple Word Prototypes (Huang et al. 2012)
Extrinsic word vector evaluation

- Extrinsic evaluation of word vectors: All subsequent tasks in this class

- One example where good word vectors should help directly: named entity recognition: finding a person, organization or location

<table>
<thead>
<tr>
<th>Model</th>
<th>Dev</th>
<th>Test</th>
<th>ACE</th>
<th>MUC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete</td>
<td>91.0</td>
<td>85.4</td>
<td>77.4</td>
<td>73.4</td>
</tr>
<tr>
<td>SVD</td>
<td>90.8</td>
<td>85.7</td>
<td>77.3</td>
<td>73.7</td>
</tr>
<tr>
<td>SVD-S</td>
<td>91.0</td>
<td>85.5</td>
<td>77.6</td>
<td>74.3</td>
</tr>
<tr>
<td>SVD-L</td>
<td>90.5</td>
<td>84.8</td>
<td>73.6</td>
<td>71.5</td>
</tr>
<tr>
<td>HPCA</td>
<td>92.6</td>
<td><strong>88.7</strong></td>
<td>81.7</td>
<td>80.7</td>
</tr>
<tr>
<td>HSMN</td>
<td>90.5</td>
<td>85.7</td>
<td>78.7</td>
<td>74.7</td>
</tr>
<tr>
<td>CW</td>
<td>92.2</td>
<td>87.4</td>
<td>81.7</td>
<td>80.2</td>
</tr>
<tr>
<td>CBOW</td>
<td>93.1</td>
<td>88.2</td>
<td>82.2</td>
<td>81.1</td>
</tr>
<tr>
<td>GloVe</td>
<td><strong>93.2</strong></td>
<td>88.3</td>
<td><strong>82.9</strong></td>
<td><strong>82.2</strong></td>
</tr>
</tbody>
</table>

- Next: How to use word vectors in neural net models!
Next lecture:

- Word classification
- With context
- Neural networks
- So amazing. Wow!